9

Message Manager

9.1 Introduction

The **message manager** provides communication and synchronization capabilities using **RTEMS** message queues. The directives provided by the message manager are:

Name	Directive Description
message_queue_create	Create a queue
message_queue_ident	Get ID of a queue
message_queue_delete	Delete a queue
message_queue_send	Put message at rear of a queue
message_queue_urgent	Put message at front of a queue
message_queue_broadcast	Broadcast N messages to a queue
message_queue_receive	Receive message from a queue
message_queue_flush	Flush all messages on a queue

9.2 Background

9.2.1 Messages

A message is a fixed length buffer where information can be stored to support communication. A message has a length of sixteen bytes. The information stored in a message is user-defined and can be actual data, pointer(s), or empty.

9.2.2 Message Queues

A **message queue** permits the passing of messages among tasks and ISRs. Message queues can contain a variable number of messages. Normally messages are sent to and received from the queue in FIFO order using the **message_queue_send** directive. However, the **message_queue_urgent** directive can be used to place messages at the head of a queue in LIFO order.

Synchronization can be accomplished when a task can wait for a message to arrive at a queue. Also, a task may poll a queue for the arrival of a message.

9.2.3 Building an Attribute Set

In general, an attribute set is built by a bitwise OR of the desired attribute components. The set of valid message queue attributes is provided in the following table:

CONSTANT	DESCRIPTION	DEFAULT
FIFO	**tasks wait by FIFO**	*
PRIORITY	**tasks wait by priority**	
NO_LIMIT	**unlimited queue size**	*
LIMIT	**limit queue size to count**	
LOCAL	**local message queue**	*
GLOBAL	**global message queue**	

An attribute listed as a default is not required to appear in the attribute list, although it is a good programming practice to specify default attributes. If all defaults are desired, the attribute **DEFAULT_ATTRIBUTES** should be specified on this call.

This example demonstrates the **attribute_set** parameter needed to create a local message queue with the task priority waiting queue discipline. The **attribute_set** parameter to the **message_queue_create** directive could be either **PRIORITY** or **LOCAL + PRIORITY**. The **attribute_set** parameter can be set to **PRIORITY** because **LOCAL** is the default for all created message queues. If a similar message queue were to be known globally, then the **attribute_set** parameter would be **GLOBAL + PRIORITY**.

9.2.4 Building an Option Set

In general, an option is built by a bitwise OR of the desired option components. The set of valid options for the **message_queue_receive** directive are listed in the following table:

CONSTANT	DESCRIPTION	DEFAULT
WAIT	task will wait for a message	*
NO_WAIT	task should not wait	

An option listed as a default is not required to appear in the option OR list, although it is a good programming practice to specify default options. If all defaults are desired, the option **DEFAULT_OPTIONS** should be specified on this call.

This example demonstrates the **option** parameter needed to poll for a message to arrive. The **option** parameter passed to the **message_queue_receive** directive should be **NO_WAIT**.

9.3 Operations

9.3.1 Creating a Message Queue

The **message_queue_create** directive creates a message queue with the user-defined name. Optionally, a limit can be placed on the number of messages allowed to be in the message queue at one time. The user may select FIFO or task priority as the method for placing waiting tasks in the task wait queue. **RTEMS** allocates a Queue Control Block (QCB) from the QCB free list to maintain the newly created queue. **RTEMS** also generates a message queue ID which is returned to the calling task.

9.3.2 Obtaining Message Queue IDs

When a message queue is created, **RTEMS** generates a unique message queue ID. The message queue ID may be obtained by either of two methods. First, as the result of an invocation of the **message_queue_create** directive, the queue ID is stored in a user provided location. Second, the queue ID may be obtained later using the **message_queue_ident** directive. The queue ID is used by other message manager directives to access this message queue.

9.3.3 Receiving a Message

The **message_queue_receive** directive attempts to retrieve a message from the specified message queue. If at least one message is in the queue, then the message is removed from the queue, copied to the caller's message buffer, and returned immediately. When messages are unavailable, one of the following situations applies:

- *By default, the calling task will wait forever for the message to arrive.*
- *Specifying the **NO_WAIT** option forces an immediate return with an error status code.*
- *Specifying a timeout limits the period the task will wait before returning with an error status.*

If the task waits for a message, then it is placed in the message queue's task wait queue in either FIFO or task priority order. All tasks waiting on a message queue are returned an error code when the message queue is deleted.

9.3.4 Sending a Message

Messages can be sent to a queue with the **message_queue_send** and **message_queue_urgent** directives. These directives work identically when tasks are waiting to receive a message. A task is removed from the task waiting queue, unblocked, and the message is copied to a waiting task's message buffer.

When no tasks are waiting at the queue, **message_queue_send** places the message at the rear of the message queue, while **message_queue_urgent** places the message at the front of the queue. The message is copied to a **RTEMS** message buffer and then placed in the message queue. Neither directive can successfully send a message to a message queue created with the **LIMIT** attribute which has a full queue of pending messages.

9.3.5 Broadcasting a Message

The **message_queue_broadcast** directive sends the same message to every task waiting on the specified message queue as an atomic operation. The message is copied to each waiting task's message buffer and each task is unblocked. The number of tasks which were unblocked is returned to the caller.

9.3.6 Deleting a Message Queue

The **message_queue_delete** directive removes a message queue from the system and frees its control block. A message queue can be deleted by any local task that knows the message queue's ID. As a result of this directive, all tasks blocked waiting to receive a message from the message queue will be readied and returned a status code which indicates that the message queue was deleted. Any subsequent references to the message queue's name and ID are invalid. Any messages waiting at the message queue are also deleted and deallocated.

9.4 Directives

This section details the message manager's directives. A subsection is dedicated to each of this manager's directives and describes the calling sequence, related constants, usage, and status codes.

9.4.1 MESSAGE_QUEUE_CREATE - Create a queue

CALLING SEQUENCE:

```
procedure message_queue_create (
   name          : in     rtems.name;
   count         : in     rtems.unsigned32;
   attribute_set : in     rtems.attribute;
   id            :    out rtems.id;
   result        :    out rtems.status_codes
);
```

DIRECTIVE STATUS CODES:

SUCCESSFUL	queue created successfully
INVALID_NAME	invalid task name
TOO_MANY	too many queues created
MP_NOT_CONFIGURED	multiprocessing not configured
TOO_MANY	too many global objects

DESCRIPTION:

This directive creates a message queue which resides on the local node with the user-defined name specified in **name**. For control and maintenance of the queue, **RTEMS** allocates and initializes a QCB. The **RTEMS**-assigned queue id, returned in **id**, is used to access the message queue.

Specifying **PRIORITY** in **attribute_set** causes tasks waiting for a message to be serviced according to task priority. When **FIFO** is specified, waiting tasks are serviced in First In-First Out order.

If **LIMIT** is specified in **attribute_set**, then a limit is fixed on the maximum number of message buffers that can be contained in the queue. Buffers are dynamically allocated from the system message buffer pool as needed. The **count** parameter is disregarded if **NO_LIMIT** is specified.

NOTES:

This directive will not cause the calling task to be preempted.

If the **LIMIT** option is selected and **count** has a value of zero, then the **message_queue_send** and **message_queue_urgent** directives will fail unless one or more tasks are already waiting on the queue.

The following message queue attribute constants are defined by **RTEMS**:

CONSTANT	DESCRIPTION	DEFAULT
FIFO	**tasks wait by FIFO**	*
PRIORITY	**tasks wait by priority**	
NO_LIMIT	**unlimited queue size**	*
LIMIT	**limit queue size to count**	
LOCAL	**local message queue**	*
GLOBAL	**global message queue**	

Message queues should not be made global unless remote tasks must interact with the created message queue. This is to avoid the system overhead incurred by the creation of a global message queue. When a global message queue is created, the message queue's name and id must be transmitted to every node in the system for insertion in the local copy of the global object table.

The total number of global objects, including message queues, is limited by the **maximum_global_objects** field in the configuration table.

9.4.2 MESSAGE_QUEUE_IDENT - Get ID of a queue

CALLING SEQUENCE:

```
procedure message_queue_ident (
   name   : in     rtems.name;
   node   : in     rtems.unsigned32;
   id     :    out rtems.id;
   result :    out rtems.status_codes
);
```

DIRECTIVE STATUS CODES:

SUCCESSFUL	queue identified successfully
INVALID_NAME	queue name not found
INVALID_NODE	invalid node id

DESCRIPTION:

This directive obtains the queue **id** associated with the queue name specified in **name**. If the queue name is not unique, then the queue id will match one of the queues with that name. However, this queue id is not guaranteed to correspond to the desired queue. The queue id is used with other message related directives to access the message queue.

NOTES:

This directive will not cause the running task to be preempted.

If **node** is **SEARCH_ALL_NODES**, all nodes are searched with the local node being searched first. All other nodes are searched with the lowest numbered node searched first.

If **node** is a valid node number which does not represent the local node, then only the message queues exported by the designated node are searched.

This directive does not generate activity on remote nodes. It accesses only the local copy of the global object table.

9.4.3 MESSAGE_QUEUE_DELETE - Delete a queue

CALLING SEQUENCE:

```
procedure message_queue_delete (
   id     : in     rtems.id;
   result :    out rtems.status_codes
);
```

DIRECTIVE STATUS CODES:

SUCCESSFUL	queue deleted successfully
INVALID_ID	invalid queue id
ILLEGAL_ON_REMOTE_OBJECT	cannot delete remote queue

DESCRIPTION:

This directive deletes the message queue specified by **id**. As a result of this directive, all tasks blocked waiting to receive a message from this queue will be readied and returned a status code which indicates that the message queue was deleted. If no tasks are waiting, but the queue contains messages, then **RTEMS** returns these message buffers back to the system message buffer pool. The QCB for this queue is reclaimed by **RTEMS**.

NOTES:

The calling task will be preempted if its preemption mode is enabled and one or more local tasks with a higher priority than the calling task are waiting on the deleted queue. The calling task will **NOT** be preempted if the tasks that are waiting are remote tasks.

The calling task does not have to be the task that created the queue, although the task and queue must reside on the same node.

When the queue is deleted, any messages in the queue are returned to the free message buffer pool. Any information stored in those messages is lost.

When a global message queue is deleted, the message queue id must be transmitted to every node in the system for deletion from the local copy of the global object table.

Proxies, used to represent remote tasks, are reclaimed when the message queue is deleted.

9.4.4 MESSAGE_QUEUE_SEND - Put message at rear of a queue

CALLING SEQUENCE:

```
procedure message_queue_send (
   id     : in     rtems.id;
   buffer : in     rtems.buffer_pointer;
   result :    out rtems.status_codes
);
```

DIRECTIVE STATUS CODES:

SUCCESSFUL	message sent successfully
INVALID_ID	invalid queue id
UNSATISFIED	out of message buffers
TOO_MANY	queue's limit has been reached

DESCRIPTION:

This directive sends the message contained in **buffer** to the queue specified by **id**. If a task is waiting at the queue, then the message is copied to the waiting task's buffer and the task is unblocked. If no tasks are waiting at the queue, then the message is copied to a message buffer which is obtained from **RTEMS**' message buffer pool. The message buffer is then placed at the rear of the queue.

NOTES:

The calling task will be preempted if it has preemption enabled and a higher priority task is unblocked as the result of this directive.

Sending a message to a global message queue which does not reside on the local node will generate a request to the remote node to post the message on the specified message queue.

If the task to be unblocked resides on a different node from the message queue, then the message is forwarded to the appropriate node, the waiting task is unblocked, and the proxy used to represent the task is reclaimed.

9.4.5 MESSAGE_QUEUE_URGENT - Put message at front of a queue

CALLING SEQUENCE:

```
procedure message_queue_urgent (
   id     : in     rtems.id;
   buffer : in     rtems.buffer_pointer;
   result :    out rtems.status_codes
);
```

DIRECTIVE STATUS CODES:

SUCCESSFUL	message sent successfully
INVALID_ID	invalid queue id
UNSATISFIED	out of message buffers
TOO_MANY	queue's limit has been reached

DESCRIPTION:

This directive sends the message contained in **buffer** to the queue specified by **id**. If a task is waiting on the queue, then the message is copied to the task's buffer and the task is unblocked. If no tasks are waiting on the queue, then the message is copied to a message buffer which is obtained from **RTEMS**' message buffer pool. The message buffer is then placed at the front of the queue.

NOTES:

The calling task will be preempted if it has preemption enabled and a higher priority task is unblocked as the result of this directive.

Sending a message to a global message queue which does not reside on the local node will generate a request telling the remote node to post the message on the specified message queue.

If the task to be unblocked resides on a different node from the message queue, then the message is forwarded to the appropriate node, the waiting task is unblocked, and the proxy used to represent the task is reclaimed.

9.4.6 MESSAGE_QUEUE_BROADCAST - Broadcast N messages to a queue

CALLING SEQUENCE:

```
procedure message_queue_broadcast (
   id     : in     rtems.id;
   buffer : in     rtems.buffer_pointer;
   count  :    out rtems.unsigned32;
   result :    out rtems.status_codes
);
```

DIRECTIVE STATUS CODES:

SUCCESSFUL	message broadcasted successfully
INVALID_ID	invalid queue id

DESCRIPTION:

This directive causes all tasks that are waiting at the queue specified by **id** to be unblocked and sent the message contained in **buffer**. Before a task is unblocked, the message in **buffer** is copied to that task's message buffer. The number of tasks that were unblocked is returned in **count**.

NOTES:

The calling task will be preempted if it has preemption enabled and a higher priority task is unblocked as the result of this directive.

The execution time of this directive is directly related to the number of tasks waiting on the message queue, although it is more efficient than the equivalent number of invocations of **message_queue_send**.

Broadcasting a message to a global message queue which does not reside on the local node will generate a request telling the remote node to broadcast the message to the specified message queue.

When a task is unblocked which resides on a different node from the message queue, a copy of the message is forwarded to the appropriate node, the waiting task is unblocked, and the proxy used to represent the task is reclaimed.

9.4.7 MESSAGE_QUEUE_RECEIVE - Receive message from a queue

CALLING SEQUENCE:

```
procedure message_queue_receive (
   id         : in     rtems.id;
   buffer     : in     rtems.buffer_pointer;
   option_set : in     rtems.option;
   timeout    : in     rtems.interval;
   result     :    out rtems.status_codes
);
```

DIRECTIVE STATUS CODES:

SUCCESSFUL	message received successfully
INVALID_ID	invalid queue id
UNSATISFIED	queue is empty
TIMEOUT	timed out waiting for message
OBJECT_WAS_DELETED	queue deleted while waiting

DESCRIPTION:

This directive receives a message from the message queue specified in **id**. The **WAIT** and **NO_WAIT** options of the **options** parameter allow the calling task to specify whether to wait for a message to become available or return immediately. For either option, if there is at least one message in the queue, then it is copied to **buffer** and this directive returns immediately with a successful return code.

If the calling task chooses to return immediately and the queue is empty, then a status code indicating this condition is returned. If the calling task chooses to wait at the message queue and the queue is empty, then the calling task is placed on the message wait queue and blocked. If the queue was created with the **PRIORITY** option specified, then the calling task is inserted into the wait queue according to its priority. But, if the queue was created with the **FIFO** option specified, then the calling task is placed at the rear of the wait queue.

A task choosing to wait at the queue can optionally specify a timeout value in the **timeout** parameter. The **timeout** parameter specifies the maximum interval to wait before the calling task desires to be unblocked. If it is set to **NO_TIMEOUT**, then the calling task will wait forever.

NOTES:

The following message receive option constants are defined by **RTEMS**:

CONSTANT	DESCRIPTION	DEFAULT
WAIT	task will wait for a message	*
NO_WAIT	task should not wait	

Receiving a message from a global message queue which does not reside on the local node will generate a request to the remote node to obtain a message from the specified message queue. If no message is available and **WAIT** was specified, then the task must be blocked until a message is posted. A proxy is allocated on the remote node to represent the task until the message is posted.

9.4.8 MESSAGE_QUEUE_FLUSH - Flush all messages on a queue

CALLING SEQUENCE:

```
procedure message_queue_flush (
   id     : in     rtems.id;
   count  :    out rtems.unsigned32;
   result :    out rtems.status_codes
);
```

DIRECTIVE STATUS CODES:

SUCCESSFUL	message received successfully
INVALID_ID	invalid queue id

DESCRIPTION:

This directive removes all pending messages from the specified queue **id**. The number of messages removed is returned in **count**. If no messages are present on the queue, **count** is set to zero.

NOTES:

Flushing all messages on a global message queue which does not reside on the local node will generate a request to the remote node to actually flush the specified message queue.